REMARKS

New evidence is submitted herewith, comprising a CD containing the 3C-SCAN software

program and a partial listing of source code implementing the statistical analysis.

The declaration of Dr. DeRosier currently of record explicitly states that each element of

each pending claim for which she was the inventor was conceived as early as November 1999

and prior to August 2000. Objective evidence of reduction to practice is submitted in the form of

the 3C-SCAN software. As the Examiner can verify by loading and executing the program, it

implements each and every claimed limitation.

The declaration of Mr. Thomas currently of record explicitly states that each element of

each pending claim for which he was the inventor was conceived as early as November 1999

and prior to August 2000. Objective evidence of reduction to practice is submitted in the form of

a partial listing of source code (since some specific statistical analysis steps in the claims are

not evident from executing the program). Both declarations state that the invention was reduced

to practice with the completion of a pre-release version of the 3C-SCAN software in August,

2000, incorporating all claimed functionality. Accordingly, the invention was conceived and

reduced to practice prior to the date of the Sherman reference.

As Mr. Thomas's declaration states, original files from the software development effort

do not exist. The only documentary evidence of the chronology of software development is the

screenshots of directory listings in Exhibit 1 to Mr. Thomas's declaration. As explained in Mr.

Thomas's declaration, this evidence shows that coding for the fully releasable version 1.0 of the

software was complete by October 11, 2000. The rules state:

> The showing of facts shall be such, in character and weight, as to establish
> reduction to practice prior to the effective date of the reference, or
> conception of the invention prior to the effective date of the reference
> coupled with due diligence from prior to said date to a subsequent
> reduction to practice or to the filing of the application. Original exhibits of
> drawings or records, or photocopies thereof, must accompany and form

part of the affidavit or declaration **or their absence must be satisfactorily explained**.

37 CFR § 1.131(b) (emphasis added).

Mr. Thomas's declaration satisfactorily explains the absence of files from the software development effort – of the two computers used, one experienced a disk crash and the files were deleted from the other. The only evidence of the software development is the screen shots of directory listings included as Exhibit 1 to Mr Thomas's declaration. These listings objectively demonstrate the creation of successive directories, each indexed by a Greek letter. Mr Thomas's declaration is evidence, in the form of sworn testimony, that each such directory corresponded to a pre-release version of the software.

Figure 3 of Exhibit 1 to Mr Thomas's declaration depicts files comprising Version 1.0 – the first releasable version – of the software. Mr Thomas's declaration is evidence, in the form of sworn testimony, that the file named *FileLev3CData.java* contained source code to Version 1.0 of the program, implementing every claimed limitation. The file listing demonstrates that this file was last modified on October 11, 2000. Accordingly, the evidence of record establishes that the claimed invention was conceived and reduced to practice by October 11, 2000, prior to the publication date of the Sherman reference. There is no evidence to the contrary, nor can the objective evidence of record be reasonably interpreted in any contrary manner.

No java source code in the Version 1.0 directory has a modification date later than October 11, 2000. As Mr Thomas's declaration states, the only file with a modification date later than the publication date of the Sherman reference is the file *3C.ve2*. This file is automatically updated by the VisualCafe development environment each time code is compiled or a build file is produced. As those of skill in the software arts know, the act of compiling code from source code to an executable form does not add any features or functionality to the code beyond that which is in the source code (other than the obvious functionality of execution under the target

operating system). The evidence thus establishes conception and reduced to practice prior to the publication date of the Sherman reference, removing that reference as prior art.

Arguendo, if a software program is not considered reduced to practice until it is compiled into executable form, the evidence demonstrates conception of all claimed limitations prior to publication of the Sherman reference and diligence in reduction to practice three days following that publication. The sworn testimony of record and the evidence of Exhibit 1 to Mr Thomas's declaration establish that the software was developed over most of 2000. Complex, sophisticated software does not write itself, and there is not a scintilla of evidence of record that Mr. Thomas and Dr. DeRosier were anything but diligent in its development. Indeed, they had every incentive to complete the software as soon as possible – no one was paying them to write it; the software development was an investment of time and energy that would only be repaid if and when the program met with commercial success.

MPEP § 2138.06 states, "A 2-day period lacking activity has been held to be fatal. *In re Mulder,* 716 F.2d 1542, 1545, 219 USPQ 189, 193 (Fed. Cir. 1983)." However, the evidence in *Mulder* overwhelmingly indicated no activity by the inventors between filing a patent application in a foreign country (which established conception) and constructive reduction to practice by filing the application in the U.S. nearly a year later. There being evidence of no activity at all, the court found there was no diligence after two days prior to filing the U.S. application, when an anticipating disclosure was published. In sharp contrast, the evidence of record in the instant case establishes substantial and continuous activity – with at least 15 pre-release versions and at least two releasable versions being developed between January and November of 2000. Indeed, the proposition that the statistical sociometric calculations Sherman teaches could have been incorporated into a 3C-SCAN program that lacked them, in three days, is preposterous (quite aside from the fact that to do so would require modifying at least one source code file).
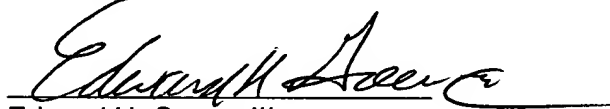
The evidence establishes conception by the inventors and reduction to practice of every claimed limitation. The evidence establishes reduction to practice by completion of source code

prior to the publication date of the Sherman reference. *Arguendo*, the evidence shows diligence in reduction to practice by the generation of executable code three days following the publication date of the Sherman reference. The lack of further evidence in the form of original software development files is satisfactorily explained, although such further evidence is not necessary.

The Sherman reference is thus removed as prior art. Applicants' response dated April 4, 2006, incorporated herein by reference, demonstrates that the SociometryPlus 2.0b reference does not teach or suggest the claimed limitations. Accordingly, all pending claims are patentably nonobvious over the art of record, and prompt allowance of all pending claims is hereby respectfully requested.

Respectfully submitted,

COATS & BENNETT, P.L.L.C.

Dated: November 14, 2006

Edward H. Green, III
Registration No.: 42,604

P.O. Box 5
Raleigh, NC 27602
Telephone: (919) 854-1844
Facsimile: (919) 854-2084

```java
import java.util.*;
import java.util.Date.*;
import java.lang.Math.*;


public class GradeLev3CData implements java.io.Serializable
{
    // jt2clean: privatize!
    public String grade_name;
    public int grade_number;
    public String form_status, de_status, da_status;
    public int number_of_rooms;
    public int number_of_students_in_grade;
    public Date last_updated;

    public boolean dataAnalyzed;
    public boolean socios[];
    public PerGroupSocioLev3CData socialPref;
    public PerGroupSocioLev3CData socialImpact;
    public PerGroupSocioLev3CData socioData[];
    public ArrayList statP, statR, statN, statC, statA, statM;

    public RoomLev3CData rooms[];
    public int currentStep;
    boolean data_changed;
    String exportSegTitle = "gradeLev0";
    //final String numFmt = " ###0.00;-###0.00";
        //java.text.DecimalFormat nf = new java.text.DecimalFormat(numFmt);
        java.text.DecimalFormat nf = FileLev3CData.nf;

    static final SocioLev3CData zeroStudentSocio = new SocioLev3CData();
    static final PerGroupSocioLev3CData zeroGroupSocio = new PerGroupSocioLev3CData();

    public GradeLev3CData ()
    {
        grade_name = "blank";
        grade_number = 0;
        currentStep = 1;
        form_status = "None";
        dataAnalyzed = false;
        data_changed = false;
        last_updated = new Date();
        socios = new boolean[FileLev3CData.MAX_SOCIOS];
        socioData = new PerGroupSocioLev3CData[FileLev3CData.MAX_SOCIOS];
        for (int i=0; i<FileLev3CData.MAX_SOCIOS; i++)
        {
            // jt2do: change to true later
            socios[i] = false;
        }
        socios[0] = true;    // jt2do: Use lookup in hash table
        socios[1] = true;

        socialPref = new PerGroupSocioLev3CData();
        socialImpact = new PerGroupSocioLev3CData();
        statP = new ArrayList();
        statR = new ArrayList();
        statN = new ArrayList();
        statC = new ArrayList();
        statA = new ArrayList();
        statM = new ArrayList();

        rooms = new RoomLev3CData[FileLev3CData.MAX_ROOMS_PER_GRADE];
    }

    public GradeLev3CData (int i)
    {
        this();
        set_grade_number(i);

    }
```

```java
import java.util.*;
import java.util.Date.*;
import java.lang.Math.*;


public class GradeLev3CData implements java.io.Serializable
{
    // jt2clean: privatize!
    public String grade_name;
    public int grade_number;
    public String form_status, de_status, da_status;
    public int number_of_rooms;
    public int number_of_students_in_grade;
    public Date last_updated;

    public boolean dataAnalyzed;
    public boolean socios[];
    public PerGroupSocioLev3CData socialPref;
    public PerGroupSocioLev3CData socialImpact;
    public PerGroupSocioLev3CData socioData[];
    public ArrayList statP, statR, statN, statC, statA, statM;

    public RoomLev3CData rooms[];
    public int currentStep;
    boolean data_changed;
    String exportSegTitle = "gradeLev0";
    //final String numFmt = " ###0.00;-###0.00";
        //java.text.DecimalFormat nf = new java.text.DecimalFormat(numFmt);
        java.text.DecimalFormat nf = FileLev3CData.nf;

    static final SocioLev3CData zeroStudentSocio = new SocioLev3CData();
    static final PerGroupSocioLev3CData zeroGroupSocio = new PerGroupSocioLev3CData();

    public GradeLev3CData ()
    {
        grade_name = "blank";
        grade_number = 0;
        currentStep = 1;
        form_status = "None";
        dataAnalyzed = false;
        data_changed = false;
        last_updated = new Date();
        socios = new boolean[FileLev3CData.MAX_SOCIOS];
        socioData = new PerGroupSocioLev3CData[FileLev3CData.MAX_SOCIOS];
        for (int i=0; i<FileLev3CData.MAX_SOCIOS; i++)
        {
            // jt2do: change to true later
            socios[i] = false;
        }
        socios[0] = true;    // jt2do: Use lookup in hash table
        socios[1] = true;

        socialPref = new PerGroupSocioLev3CData();
        socialImpact = new PerGroupSocioLev3CData();
        statP = new ArrayList();
        statR = new ArrayList();
        statN = new ArrayList();
        statC = new ArrayList();
        statA = new ArrayList();
        statM = new ArrayList();

        rooms = new RoomLev3CData[FileLev3CData.MAX_ROOMS_PER_GRADE];
    }

    public GradeLev3CData (int i)
    {
        this();
        set_grade_number(i);

    }
```

```java
public void add_room(int whichroom)
{
    rooms[whichroom] = new RoomLev3CData();
}

public void addSocio(int idx)
{
    //jt2do: We set all socios true as a convenience
    //         but delay allocation until DataEntry time
    if ( socioData[idx] == null )
        socioData[idx] = new PerGroupSocioLev3CData();
    // System.out.println("addSocio " + Integer.toString(idx) );
    //Now add socio to each room, and each student within each room
    for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++)
        if (!get_is_room_empty(j))
            rooms[j].addSocio(idx);
}

public void set_grade_name(String s)
{
    grade_name = s;
}

public String get_grade_name()
{
    if (grade_number==0)
        return("K");
    else
        return Integer.toString(grade_number);
}

public void set_grade_number(int i)
{
    grade_number = i;
}
public int get_grade_number()
{
    return grade_number;
}

public boolean get_is_room_empty(int i)
{
    if (rooms[i]==null) return true;
    else {
        return (rooms[i].is_empty());
    }
}

public RoomLev3CData get_room(int i)
{
    // jt2do: test for out of bounds, throw exception on empty?
    return rooms[i];
}
public void set_room(int i, RoomLev3CData thisroom)
{
    // jt2do: is a "this" required?
    rooms[i] = thisroom;
}

public int get_number_of_rooms()
{
    int i=0;
    for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {

        if (!this.get_is_room_empty(j))
            i++;
    }
    return i;
}

public void set_number_of_rooms(int i)
```

```java
{
    // Delete this soon
    System.exit(-1);
}

public int get_number_of_students_in_grade()
{
    return number_of_students_in_grade;
}
public void set_number_of_students_in_grade(int i)
{
    number_of_students_in_grade = i;
}


public boolean impactActive()
{
    // Can only compute social impact and social preference if socios 0 & 1 are true
    return (socios[0]==true & socios[1]==true);
}

public StudentLev3CData findStudentInGrade(String name)
{
    StudentLev3CData tempName=null;
    for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
        if (!this.get_is_room_empty(j)) {
            tempName = this.rooms[j].findStudentInRoom(name);
        }
        // Once we find the dude in one room, let's stop looking
        if (tempName!=null) break;
    }
    return tempName;
}

public RoomLev3CData findRoomContainingStudent(String name)
{
    StudentLev3CData tempName=null;
    for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
        if (!this.get_is_room_empty(j)) {
            tempName = this.rooms[j].findStudentInRoom(name);
            if(tempName!=null) return this.rooms[j];
        }
    }
    return null;
}




public RoomLev3CData findRoomTaughtBy(String teacherName)
{
    for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
        if(!this.get_is_room_empty(j))
            if(this.rooms[j].isTaughtBy(teacherName))
                return this.rooms[j];  // found a match
    }  // loop over rooms
    return null;
}

public int get_number_of_socio_pages()
{
    int n=0;

    // jt: very kludgey...
    // in fact, it doesn't work

    for (int i=0; i<FileLev3CData.MAX_SOCIOS; ++i) {
        // Special cases...we never print these next two
        if (!FileLev3CData.INTERNAL_SOCIO_NAMES[i].equals("$BFRIEND"))
            if (!FileLev3CData.INTERNAL_SOCIO_NAMES[i].equals("$VBFRIEND"))
                if (socios[i]==true) n++;
    }
```

```java
        return n;
    }


public int get_nth_printable_socidx(int n)
{
    for (int i=0; i<FileLev3CData.MAX_SOCIOS; ++i) {
        // Special cases...we never print these next two
        if (!FileLev3CData.INTERNAL_SOCIO_NAMES[i].equals("$BFRIEND"))
            if (!FileLev3CData.INTERNAL_SOCIO_NAMES[i].equals("$VBFRIEND"))
                if (socios[i]==true) {
                    if (n==0) return i;
                    --n; // one less active socidx
                }
    }

        return -1;
    }
public boolean get_data_changed ()
{
    return data_changed;
}

public void set_data_changed (boolean bit)
{
    // Just sets highest level bit
    data_changed = bit;
}


public void clearSummarySocios()
{
    dataAnalyzed = false;
    for (int i=0; i<FileLev3CData.MAX_SOCIOS; ++i)
        if (socioData[i] != null) socioData[i].clear();

    socialPref.clear();
    socialImpact.clear();
    statP.clear();
    statR.clear();
    statN.clear();
    statC.clear();
    statA.clear();
    statM.clear();

    for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
        if (!this.get_is_room_empty(j)) {
            this.rooms[j].clearRoomSocios();
        }
    }
}


public void clearAllSocios()
{
    clearSummarySocios();
    for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
        if (!this.get_is_room_empty(j)) {
            this.rooms[j].clearStudentSocios();
        }
    }
}

public PerGroupSocioLev3CData getSocio(int i)
{
    if (socioData[i] != null)
        return socioData[i];
    else
        return GradeLev3CData.zeroGroupSocio;
}
```

```java
    public PerGroupSocioLev3CData getSocio(String s)
    {   .                        .
        // Convert String socio title to name
        int index =' FileLev3CData.convertToSocioIndex(s);
        if (index!=-1)
          return (this.getSocio(index));
        else
          return GradeLev3CData.zeroGroupSocio;
    }

    public void exportAllStudents(Exportable3CData xpData)
    {
        xpData.appendField(exportSegTitle, "gradeName", this.get_grade_name());
        xpData.appendField(exportSegTitle, "gradeNumber", Integer.toString(grade_number));

        // for (int i=0; i<FileLev3CData.MAX_SOCIOS; ++i)
        //    if (socioData[i] != null) socioData[i].exportSocioData(xpData, FileLev3CData.
SOCIO_NAMES[i]);

        // socialPref.exportSocioData(xpData, "SocialPref");
        // socialImpact.exportSocioData(xpData, "SocialImpact");;

        for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
            if (!this.get_is_room_empty(j)) {
                this.rooms[j].exportRoomData(xpData);
            }
        }
        // System.out.println("out of grade");
    }


  public void analyzeGrade()
        {
                int tempInt=0;
                double tempDouble=0.0;

            int socidx=0;
        StudentLev3CData student;
        String txt;
        double strengthDelta = 0.0;
        int changeCount = 0;

                    TreeSet strengthPop, strengthRej, strengthCon, strengthNeg;
            strengthPop = new TreeSet();
            strengthRej = new TreeSet();
            strengthCon = new TreeSet();
            strengthNeg = new TreeSet();

        // jtadd4prob: {
        final String zFmt = " #.00;-#.00";
        java.text.DecimalFormat zf = new java.text.DecimalFormat(zFmt);

        ArrayList zSPgt1, zLMgt0, zLLlt0;
        ArrayList zSPltneg1, zLLgt0;
        ArrayList zSIltneg1, zLMlt0;
        ArrayList zSIgt1;
        // jtadd4prob: }

        double covPopAB, covPopBC, covPopAC;
        double covRejAB, covRejBC, covRejAC;
        double covNegAB, covNegBC, covNegAC;
        double covConAB, covConBC, covConAC;
        double covAveAB, covUncAB;

        covPopAB = covPopBC = covPopAC = 0;
        covRejAB = covRejBC = covRejAC = 0;
        covNegAB = covNegBC = covNegAC = 0;
        covConAB = covConBC = covConAC = 0;
        covAveAB = covUncAB = 0;

        zSPgt1 = new ArrayList();
```

```
        . zLMgt0 = new ArrayList();
        zLLlt0 = new ArrayList();
        zSPltneg1 = new ArrayList();
        zLLgt0`= new ArrayList();
        zSIltneg1 = new ArrayList();
        zLMlt0 = new ArrayList();
        zSIgt1 = new ArrayList();
        // jtadd4prob: }


//      if (dataAnalyzed & !TopLevel.exportStatsForSRCD) return;
        if (dataAnalyzed) return;

// Changing internal data
        set_data_changed(true);

/*------*
 *
 *   Phase 1: Compute raw, mean, and SD for each socio in grade
 *
 *------*/
        for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
        if (!get_is_room_empty(j)) {
            //  System.out.println("  Room: " + j);
          RoomLev3CData room;
          int num_students;

          room = get_room(j);
          num_students = room.get_number_of_students();

          // Loop over students
          for(int k=0; k < num_students; k++) {
              student = room.get_student(k);
            // System.out.println("  Student: " + k);

            /*
              if(k==0) {
                room.findStudentInRoom(student.get_student_name());
                room.findStudentInRoom(student.get_student_name_no_prefix());
                room.findStudentInRoom(student.getStudentFormCode());
              } */

              for (socidx=0; socidx<FileLev3CData.MAX_SOCIOS; ++socidx) {
                  if (this.socios[socidx]==true)
                  {
                tempInt = student.getSocio(socidx).getRawCount();
                this.socioData[socidx].addCount(tempInt);
                  // room.socioData[socidx].addCount(tempInt);
              } // This grade has this socio
          } // Loop - socios
        } // Loop - students

      } // Room is not empty
} // Loop over rooms

// System.out.println("  analyze - starting phase 2");

 /*------*
  *
  *   Phase 2: Compile individual Z scores, SPref, SImpact
  *
  *------*/
                for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
        if (!get_is_room_empty(j)) {
          RoomLev3CData room;
          int num_students;

          room = get_room(j);
          num_students = room.get_number_of_students();

          // Loop over students
```

```java
            for(int k=0; k < num_students; k++) {
                student = room.get_student(k);
                for (socidx=0; socidx<FileLev3CData.MAX_SOCIOS; ++socidx) {
                    if (this.socios[socidx]==true)
                    {
                        //       System.out.println("         setMyZscore " + socidx);
                        tempInt = student.getSocio(socidx).getRawCount();
                        student.getSocio(socidx).setMyZscore(socioData[socidx].RawToZscore
(tempInt));

                        tempDouble = student.getSocio(socidx).getMyZscore();
                        if (tempDouble > 1)
                        {
                            // This is extraordinary, going to add it to some tier
                            if (tempDouble > 3) {
                                this.socioData[socidx].incrTier3();
                                room.getSocio(socidx).incrTier3();
                            }
                            else if (tempDouble > 2) {
                                this.socioData[socidx].incrTier2();
                                room.getSocio(socidx).incrTier2();
                            }
                            else {
                                this.socioData[socidx].incrTier1();
                                room.getSocio(socidx).incrTier1();
                            }
                        }
                        // Get recip counts
                        tempInt = student.getSocio(socidx).getReciprocalNominations().size();

                        // At a group level, we track the number of students who have recip pairs
                        if(tempInt>0) {
                            room.getSocio(socidx).incrNumReciprocalPairs(1);
                            this.socioData[socidx].incrNumReciprocalPairs(1);
                        }

                        // Get self-nom counts (incr by one)
                        if (student.getSocio(socidx).choseSelf()) {
                            // System.out.println("incrementting socidx:" + socidx + " for " +
student.get_student_name());
                            this.socioData[socidx].incrNumSelfNominations();
                            room.getSocio(socidx).incrNumSelfNominations();
                        }
                    } // This grade has this socio
                } // Loop - socios

                if (impactActive()) {
                    // jtadd4prob: {
                    // Social Pref = ZLikedMost - ZLikedLeast
                    double zzLM, zzLL;
                    zzLM = student.getSocio("$LM").getMyZscore();
                    zzLL = student.getSocio("$LL").getMyZscore();
                    String name = student.get_student_name();

                    student.mySocialPref = zzLM - zzLL;

                    // Social Pref = ZLikedMost - ZLikedLeast
                    //student.mySocialPref = student.getSocio("$LM").getMyZscore() - student.
getSocio("$LL").getMyZscore();

                    // Add to grade level so we can later standardize
                    this.socialPref.addCount(student.mySocialPref);


                    // Social Impact = ZLikedMost + ZLikedLeast
                    student.mySocialImpact = zzLM + zzLL;

                    //student.mySocialImpact = student.getSocio("$LM").getMyZscore() + student.
getSocio("$LL").getMyZscore();

                    // Add to grade level so we can later standardize
                    this.socialImpact.addCount(student.mySocialImpact);
```

```java
            if (student.mySocialPref > 1) zSPgt1.add(name);
            if.(student.mySocialPref < -1) zSPltneg1.add(name);

            if (student.mySocialImpact > 1) zSIgt1.add(name);
            if (student.mySocialImpact < -1) zSIltneg1.add(name);

            if (zzLM > 0) zLMgt0.add(name);
            if (zzLM < 0) zLMlt0.add(name);

            if (zzLL > 0) zLLgt0.add(name);
            if (zzLL < 0) zLLlt0.add(name);

            // compute covariances
            /*

            PzSPPos1 = cdf1(ZSPREF1 - 1.0);
            PzSPNeg1 = cdf1(-1.0 - ZSPREF1);
            PzSIPos1 = cdf1(ZSIMP1 - 1.0);
            PzSINeg1 = cdf1(-1.0 - ZSIMP1);
            PzLikePos = cdf1(ZLIKE1);
            PzLikeNeg = 1 - PzLikePos;

            PzDLikePos = cdf1(ZDLIKE1);
            PzDLikeNeg = 1 - PzDLikePos;
            P(Popular) =        ( PzSPPos1 * PzLikePos * PzDLikeNeg) ];
            P(Rejected) =       ( PzSPNeg1 * PzLikeNeg * PzDLikePos) ];
            P(Neglected) =      ( PzSINeg1 * PzLikeNeg * PzDLikeNeg) ];
            P(Controversial) = ( PzSIPos1 * PzLikePos * PzLDikePos) ];
            */

        // Compute covariances of constituent componets of status designations
        covPopAB = covPopAB + ( cdf1((student.mySocialPref - 1.0)) * cdf1(zzLM) );
        covPopBC = covPopBC + ( cdf1(zzLM) * (1 - cdf1(zzLL)) );
        covPopAC = covPopAC + ( cdf1((student.mySocialPref - 1.0)) * (1 - cdf1(zzLL)) );

        covRejAB = covRejAB + ( cdf1((-1.0 - student.mySocialPref)) * (1 - cdf1(zzLM))  );
        covRejBC = covRejBC + ( (1 - cdf1(zzLM)) * cdf1(zzLL) );
        covRejAC = covRejAC + ( cdf1((-1.0 - student.mySocialPref)) * (cdf1(zzLL)) );

        covNegAB = covNegAB + ( cdf1((-1.0 - student.mySocialImpact)) * (1 - cdf1(zzLM)));
        covNegBC = covNegBC + ( (1 - cdf1(zzLM)) * (1 - cdf1(zzLL)));
        covNegAC = covNegAC + (cdf1((-1.0 - student.mySocialImpact)) * (1 - cdf1(zzLL)));

        covConAB = covConAB + (cdf1((student.mySocialImpact - 1.0)) * cdf1(zzLM));
        covConBC = covConBC + ( cdf1(zzLM) *  cdf1(zzLL) );
        covConAC = covConAC + (cdf1((student.mySocialImpact - 1.0)) * cdf1(zzLL));

        //              PzSIAverage = P(-.5<zSI<.5) = (1-cdf(zSI -.5)) * (1-cdf(-.5-
zSI))

        covAveAB = covAveAB + (
        (1 - cdf1((student.mySocialImpact - 0.5))) * (1 - cdf1((0.5 - student.
mySocialImpact))) *
        (1 - cdf1((student.mySocialPref - 0.5))) * (1 - cdf1((0.5 - student.mySocialPref)))
         );

        covUncAB = covUncAB + (
                        (1 - cdf1((student.mySocialImpact - 1.0))) * (1 - cdf1((1.0
- student.mySocialImpact))) *
                        (1 - cdf1((student.mySocialPref - 1.0))) * (1 - cdf1((1.0 -
student.mySocialPref)))
                        );

            // jtadd4prob: }

        } // impactActive

        } // Loop - students
    } // Room is not empty
} // Loop over rooms
```

```java
        // System.out.println("  analyze - starting phase 3");
        /*-----*
         *
         * Phase 3: Compute gradelevel intersection sets and conditional probability
coefficients
         *          to be used in individual sociostatus probabilities calculations.  We can
only
         *          do this if we have LM and LL voting in effect, of course.
         *-----*/


        if (impactActive()) {

                /*
                Conditional probability lesson...

                P(B|A) = P(B & A) / P(B)
                P(C|A&B) = P(C & (A & B)) / P(C)
                */

                // Complete covariance calculations

                covPopAB = covPopAB / this.number_of_students_in_grade;
                covPopBC = covPopBC / this.number_of_students_in_grade;
                covPopAC = covPopAC / this.number_of_students_in_grade;

                covRejAB = covRejAB / this.number_of_students_in_grade;
                covRejBC = covRejBC / this.number_of_students_in_grade;
                covRejAC = covRejAC / this.number_of_students_in_grade;

                covNegAB = covNegAB / this.number_of_students_in_grade;
                covNegBC = covNegBC / this.number_of_students_in_grade;
                covNegAC = covNegAC / this.number_of_students_in_grade;

                covConAB = covConAB / this.number_of_students_in_grade;
                covConBC = covConBC / this.number_of_students_in_grade;
                covConAC = covConAC / this.number_of_students_in_grade;

                covAveAB = covAveAB / this.number_of_students_in_grade;

                covUncAB = covUncAB / this.number_of_students_in_grade;


        } // impactActive

/*
    System.out.println("pPopAB: " + pPopAB);
    System.out.println("pPopABC: " + pPopABC);
    System.out.println("pRejAB: " + pRejAB);
    System.out.println("pRejABC: " + pRejABC);
    System.out.println("pNegAB: " + pNegAB);
    System.out.println("pNegABC: " + pNegABC);
    System.out.println("pConAB: " + pConAB);
    System.out.println("pConABC: " + pConABC);

    System.out.println("covPopAB: " + covPopAB);
    System.out.println("covPopBC: " + covPopBC);
    System.out.println("covPopAC: " + covPopAC);
    System.out.println("covRejAB: " + covRejAB);
    System.out.println("covRejBC: " + covRejBC);
    System.out.println("covRejAC: " + covRejAC);
    System.out.println("covNegAB: " + covNegAB);
    System.out.println("covNegBC: " + covNegBC);
    System.out.println("covNegAC: " + covNegAC);
    System.out.println("covConAB: " + covConAB);
    System.out.println("covConBC: " + covConBC);
    System.out.println("covConAC: " + covConAC);

    System.out.println("covAveAB: " + covAveAB);
    System.out.println("covUncAB: " + covUncAB);
```

```java
*/


/*------*
 *
 *  Phase 4: Compute Individual Z scores and Status
 *
 *------*/
// jt2dorsn: Ensure If we have socios 0&1

if (impactActive()) {


    for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
        if (!get_is_room_empty(j)) {
            RoomLev3CData room;
            int num_students;

            room = get_room(j);
            num_students = room.get_number_of_students();
            //  System.out.println("     Room: " + room.get_teacher_name());


            // Loop over students
            for(int k=0; k < num_students; k++) {
                student = room.get_student(k);
                String name = student.get_student_name();
                // jt2do: Ensure If we have socios 0&1
                student.mySocialPrefZ = this.socialPref.RawToZscore(student.mySocialPref);
                student.mySocialImpactZ = this.socialImpact.RawToZscore(student.
mySocialImpact);
                // Determine Status
                double ZSPREF1 = student.mySocialPrefZ;
                double ZSIMP1 = student.mySocialImpactZ;
                double ZLIKE1 = student.getSocio("$LM").getMyZscore();
                double ZDLIKE1 = student.getSocio("$LL").getMyZscore();
                int LMOST1 = student.getSocio("$LM").getRawCount();
        /*
        System.out.println("       ZSPREF1: " + ZSPREF1);
        System.out.println("       ZSPREF1: " + ZSPREF1);
        System.out.println("       ZSIMP1: " + ZSIMP1);
        System.out.println("       ZLIKE1: " + ZLIKE1);
        System.out.println("       LMOST1: " + LMOST1);
        */

                /* Old Algorithm (decommissioned)
                if ((ZSPREF1 > 1.0) & (ZDLIKE1 < 0) & (ZLIKE1 > 0))
                {
                    student.mySocioStatus = "Popular";
                    statP.add(name);
                }
                else if ((ZSPREF1 < -1.0) & (ZLIKE1 < 0) & (ZDLIKE1 > 0))
                {
                    student.mySocioStatus = "Rejected";
                    statR.add(name);
                }
                else if ((ZSIMP1 < -1.0) & (LMOST1 == 0))
                {
                    student.mySocioStatus = "Neglected";
                    statN.add(name);
                }
                else if ((ZSIMP1 > 1.0) & (ZLIKE1 > 0) & (ZDLIKE1 > 0))
                {
                    student.mySocioStatus = "Controversial";
                    statC.add(name);
                }
                else if ((-0.5 < ZSIMP1) & (ZSIMP1 < 0.5) & (-0.5 < ZSPREF1) & (ZSPREF1 < 0.5))
                {
                    student.mySocioStatus = "Average";
                    statA.add(name);
                }
```

```java
    else
    {
        student.mySocioStatus = "Unclassified";
        StatM.add(name);
    }

    student.mySocioStatus = student.mySocioStatus + "(old algorithm),  ";
    */

    //------------------New algorithm (early 2001)

    student.mySocioStatus = ""; // Reset in case we came through before
    if ((ZSPREF1 > 1.0) & (ZDLIKE1 < 0) & (ZLIKE1 > 0))
    {
        student.mySocioStatus = "Popular";
        this.statP.add(name);
        room.statP.add(name);
    }
    else if ((ZSPREF1 < -1.0) & (ZLIKE1 < 0) & (ZDLIKE1 > 0))
    {
        student.mySocioStatus = "Rejected";
        this.statR.add(name);
        room.statR.add(name);
    }
    else if ((ZSIMP1 < -1.0) & (ZLIKE1 < 0) & (ZDLIKE1 < 0))
    {
        student.mySocioStatus = "Neglected";
        this.statN.add(name);
        room.statN.add(name);
    }
    else if ((ZSIMP1 > 1.0) & (ZLIKE1 > 0) & (ZDLIKE1 > 0))
    {
        student.mySocioStatus =  "Controversial";
        this.statC.add(name);
        room.statC.add(name);
    }
    else if ((-0.5 < ZSIMP1) & (ZSIMP1 < 0.5) & (-0.5 < ZSPREF1) & (ZSPREF1 < 0.5))
    {
        student.mySocioStatus = "Average";
        this.statA.add(name);
        room.statA.add(name);
    }
    else
    {
        student.mySocioStatus = "Unclassified";
        this.statM.add(name);
        room.statM.add(name);
    }


    // Adding new stuff for Probabilities here and appending
    // to mySocioStatus since it was conveniently defined
    // as a string (thx, jt)
    //
    /*
        Definitions:

        P(Popular) =        [ P(zSP>1)  * P(zLike>0) * P(zDLike<0)]
        P(Rejected) =       [ P(zSP<-1) * P(zLike<0) * P(zDLike>0)]
        P(Neglected) =      [ P(zSI<-1) * P(zLike<0) * P(zDlike<0)]
        P(Controversial) = [ P(zSI>1)   * P(zLike>0) * P(zDlike>0 ]
        P(Avergage) =       [ P(-.5<zSP<.5) * P(-5.<zSI<.5)]
        P(Unclassified) =   [ P(-1<zSI<1) * P(-1<zSP<1)] - P(Average);

        Breaking it down for SP...
        PzSPPos1 = P(zSP) > 1 = cdf (zSP -1)
        PzSPNeg1 = P(zSP) < -1 = cdf (-1-zSP)
        PzSPNominal = P(-1<zSP<1) = 1-PzSPPos1 * 1-PzSPNeg1
        PzSPAverage = P(-.5<zSP<.5) = (1-cdf(zSP -.5)) * (1-cdf(-.5-zSP))

        For SI...
```

```
        PzSIPos1 = P(zSI) > 1 = cdf (zSI -1)
        PzSINeg1 = P(zSI) < -1 = cdf (-1-zSI)
        PzSINominal = P(-1<zSI<1) = 1-PzSIPos1 * 1-PzSINeg1
        PzSIAverage = P(-.5<zSI<.5) = (1-cdf(zSI -.5)) * (1-cdf(-.5-zSI))


        For ZLike
        PzLikePos = P(zLike > 0);
        PzLikeNeg = 1 - PzLikePos;


        For ZDlike
        PzDLikePos = P(ZDlike > 0);
        PzDlikeNeg = 1 - PzDlikePos;


        So...
        P(Popular) =        ( PzSPPos1 * PzLikePos * PzDLikeNeg) ];
        P(Rejected) =       ( PzSPNeg1 * PzLikeNeg * PzDLikePos) ];
        P(Neglected) =      ( PzSINeg1 * PzLikeNeg * PzDLikeNeg) ];
        P(Controversial) = ( PzSIPos1 * PzLikePos * PzLDikePos) ];
        P(Avergage) =       ( PzSPAverage * PzSIAverage );
        P(Unclassified) =   ( PzSINominal * PzSPNominal) - P(Average)


        */
        double PzSPPos1, PzSPNeg1, PzSPNominal, PzSPAverage;
        double PzSIPos1, PzSINeg1, PzSINominal, PzSIAverage;
        double PzLikePos, PzLikeNeg, PzDLikePos, PzDLikeNeg;
        // System.out.println("              analyze - starting probs");

        PzSPPos1 = cdf1(ZSPREF1 - 1.0);
        PzSPNeg1 = cdf1(-1.0 - ZSPREF1);
        PzSPNominal = (1-PzSPPos1) * (1-PzSPNeg1);
        // old way, which was wrong: PzSPAverage = (1-cdf1(ZSPREF1 -.5)) * (1-cdf1(-.
5-ZSPREF1)));

        PzSPAverage = (1-cdf1(ZSPREF1 -.5)) * cdf1( .5 + ZSPREF1);

        PzSIPos1 = cdf1(ZSIMP1 - 1.0);
        PzSINeg1 = cdf1(-1.0 - ZSIMP1);
        PzSINominal = (1-PzSIPos1) * (1-PzSINeg1);
        // old way, which was wrong: PzSIAverage = (1-cdf1(ZSIMP1 -.5)) * (1-cdf1(-.5-
ZSIMP1)));

        PzSIAverage = (1-cdf1(ZSIMP1 -.5)) * cdf1(+.5+ZSIMP1);
        PzLikePos = cdf1(ZLIKE1);
        PzLikeNeg = 1 - PzLikePos;

        PzDLikePos = cdf1(ZDLIKE1);
        PzDLikeNeg = 1 - PzDLikePos;

        // Compute individual probabilities - flawed because of overlapping
dependencies
        //student.pPop = Float.valueOf(nf.format(PzSPPos1 * PzLikePos * PzDLikeNeg)).
floatValue();
        //student.pRej = Float.valueOf(nf.format(PzSPNeg1 * PzLikeNeg * PzDLikePos)).
floatValue();
        //student.pNeg = Float.valueOf(nf.format(PzSINeg1 * PzLikeNeg * PzDLikeNeg)).
floatValue();
        //student.pCon = Float.valueOf(nf.format(PzSIPos1 * PzLikePos * PzDLikePos)).
floatValue();
        //student.pAve = Float.valueOf(nf.format(PzSPAverage * PzSIAverage)).
floatValue();
        //student.pUnc = Float.valueOf(nf.format((PzSINominal * PzSPNominal) -
(PzSPAverage * PzSIAverage))).floatValue();

        // Better computation ahead for individual probabilities...
                            /*
                double pPopAB, pPopABC;
                double pRejAB, pRejABC;
                double pNegAB, pNegABC;
                double pConAB, pConABC;

                Conditional probability lesson...
```

```
                            P(B|A) = P(B & A) / P(B)
                            P(C|A&B) = P(C & (A & B)) / P(C)

                            But how does that change our P(A) * P(B) * P(C)?
                            */


/*
                            PzSPPos1 = PzSPPos1 * PzSPPos1;
                            PzLikePos = PzLikePos * PzLikePos;
                            PzDLikeNeg = PzDLikeNeg * PzDLikeNeg;

                            PzSPNeg1 = PzSPNeg1 * PzSPNeg1;
                            PzLikeNeg = PzLikeNeg * PzLikeNeg;
                            PzDLikePos = PzDLikePos * PzDLikePos;

                            PzSINeg1 = PzSINeg1 * PzSINeg1;
                            PzLikeNeg = PzLikeNeg * PzLikeNeg;
                            PzDLikeNeg = PzDLikeNeg * PzDLikeNeg;

                            PzSIPos1 = PzSIPos1 * PzSIPos1;
                            PzLikePos = PzLikePos * PzLikePos;
                            PzDLikePos = PzDLikePos * PzDLikePos;
*/

                    // Revised probaability calcs, using covariance to control for interdepence


            student.pPop = Float.valueOf(nf.format(
                        (PzSPPos1 * PzLikePos * PzDLikeNeg) +
                2 * (covPopAB * covPopBC * covPopAC) -
                        (PzLikePos * covPopAC * covPopAC) -
                        (PzSPPos1 * covPopBC * covPopBC) -
                        (PzDLikeNeg * covPopAB * covPopAB))).floatValue();

            student.pRej = Float.valueOf(nf.format(
                        (PzSPNeg1 * PzLikeNeg * PzDLikePos) +
                2 * (covRejAB * covRejBC * covRejAC) -
                        (PzLikeNeg * covRejAC * covRejAC) -
                        (PzSPNeg1 * covRejBC * covRejBC) -
                        (PzDLikePos * covRejAB * covRejAB))).floatValue();

            student.pNeg = Float.valueOf(nf.format(
                        (PzSINeg1 * PzLikeNeg * PzDLikeNeg) +
                2 * (covNegAB * covNegBC * covNegAC) -
                        (PzLikeNeg * covNegAC * covNegAC) -
                        (PzSINeg1 * covNegBC * covNegBC) -
                        (PzDLikeNeg * covNegAB * covNegAB))).floatValue();

            student.pCon = Float.valueOf(nf.format(
                        (PzSIPos1 * PzLikePos * PzDLikePos) +
                2 * (covConAB * covConBC * covConAC) -
                        (PzLikePos * covConAC * covConAC) -
                        (PzSIPos1 * covConBC * covConBC) -
                        (PzDLikePos * covConAB * covConAB))).floatValue();

                        student.pAve = Float.valueOf(nf.format(PzSPAverage *
PzSIAverage)).floatValue();
                        student.pUnc = Float.valueOf(nf.format((PzSINominal *
PzSPNominal) - (PzSPAverage * PzSIAverage))).floatValue();

                    //      System.out.println("   --- Raw Ave (univariate) --- " +
student.pAve + " Unclassified --- " + student.pUnc);

                        student.pAve = Float.valueOf(nf.format(
                        (PzSPAverage * PzSIAverage) +
                        (covAveAB * covAveAB) -
                        (covAveAB * PzSPAverage) -
                        (covAveAB * PzSIAverage)
```

```
                                                    )).floatValue();

                    student.pUnc = Float.valueOf(nf.format(
                        (PzSINominal * PzSPNominal) +
                        (covUncAB * covUncAB) -
                        (covUncAB * PzSINominal) -
                        (covUncAB * PzSPNominal) -

                        ((PzSPAverage * PzSIAverage) +
                        (covAveAB * covAveAB) -
                        (covAveAB * PzSPAverage) -
                        (covAveAB * PzSIAverage))
                                )).floatValue();


        //                 System.out.println("    ---------- New ---------- " + student.
pAve + " Unclassified --- " + student.pUnc);

                    // jttemp
                    /*
                    System.out.println("    --- Raw Pop (new) --- " + student.pPop);
                    System.out.println("    --- Raw Rej (new) --- " + student.pRej);
                    System.out.println("    --- Raw Con (new) --- " + student.pCon);
                    System.out.println("    --- Raw Neg (new) --- " + student.pNeg);
                    System.out.println("    --- Raw Ave (new) --- " + student.pAve);
                    System.out.println("    --- Raw Unc (new) --- " + student.pUnc);
                    */

                    if (student.pPop < 0) student.pPop = 0;
                    if (student.pRej < 0) student.pRej = 0;
                    if (student.pCon < 0) student.pCon = 0;
                    if (student.pNeg < 0) student.pNeg = 0;
                    if (student.pAve < 0) student.pAve = 0;
                    if (student.pUnc < 0) student.pUnc = 0;


                    if (student.pPop > 1) student.pPop = 1;
                    if (student.pRej > 1) student.pRej = 1;
                    if (student.pCon > 1) student.pCon = 1;
                    if (student.pNeg > 1) student.pNeg = 1;
                    if (student.pAve > 1) student.pAve = 1;
                    if (student.pUnc > 1) student.pUnc = 1;

            float totalP = student.pPop +
                        student.pRej +
                        student.pCon +
                        student.pNeg +
                        student.pAve +
                        student.pUnc;
//  System.out.println("            Prob calc 2");
            // Compute relative probabiities or
            // affinity coeffecients - they have to be something
            student.pPop = Float.valueOf(nf.format(student.pPop/totalP)).floatValue();
            student.pRej = Float.valueOf(nf.format(student.pRej/totalP)).floatValue();
            student.pCon = Float.valueOf(nf.format(student.pCon/totalP)).floatValue();
            student.pNeg = Float.valueOf(nf.format(student.pNeg/totalP)).floatValue();
            student.pAve = Float.valueOf(nf.format(student.pAve/totalP)).floatValue();
            student.pUnc = Float.valueOf(nf.format(student.pUnc/totalP)).floatValue();


            // Convert probability of designated status into a strength score
            // jtwbn: Break down the Unclassifieds by quadrant, apply corresponding
            //        pole score
            // System.out.println("              Strength calc");
            Float strength;

            // CHRIS:  Here I am building up sorted lists of the
            //         strength scores within each status group.
            //         Note that we are within a loop over student
            //         here and we will need to let that loop run
            //         through to build these sorted lists all the
```

```java
//              way.  Unfortunately, the TreeSet collection
//              is expecting objects and I am getting type
//              mismatch compiler errors throughout because
//.             I have not done all the right conversions.
//              This may not be the best way to do it.
        if (student.mySocioStatus.equals("Popular"))
            strengthPop.add( new Float(student.pPop) );
        else if (student.mySocioStatus.equals("Rejected"))
            strengthRej.add( new Float( student.pRej ));
        else if (student.mySocioStatus.equals("Controversial"))
            strengthCon.add( new Float(student.pCon));
        else if (student.mySocioStatus.equals("Neglected"))
            strengthNeg.add( new Float(student.pNeg));
//          else if (student.mySocioStatus.equals("Average"))
//              strength = student.pAve;
//          else
//              strength= -1.0;

            if (student.mySocioStatus.equals("Unclassified"))  {
        student.statusStrength= 0;
        if (student.pPop > student.pRej)
            if (student.pPop > student.pCon)
                if (student.pPop > student.pNeg)
                    student.myUnclassifiedBias = "Popular Bias";
                else
                    student.myUnclassifiedBias = "Neglected Bias";
            else
                if (student.pCon > student.pNeg)
                    student.myUnclassifiedBias = "Controversial Bias";
                else
                    student.myUnclassifiedBias = "Neglected Bias";
        else
            if (student.pRej > student.pCon)
                if (student.pRej > student.pNeg)
                    student.myUnclassifiedBias = "Rejected Bias";
                else
                    student.myUnclassifiedBias = "Neglected Bias";
            else
                if (student.pCon > student.pNeg)
                    student.myUnclassifiedBias = "Controversial Bias";
                else
                    student.myUnclassifiedBias = "Neglected Bias";
        }
        else {
          student.myUnclassifiedBias = "";
            // CHRIS: I left this in for reference, but it
            //        should probably be removed when
            //        the next loop works.
//              if (strength>= .75)
//                  student.statusStrength= 3;
//              else if (strength>= .50)
//                  student.statusStrength= 2;
//              else student.statusStrength= 1;
            }
    } // Loop - students
  } // Room is not empty
 } // Loop over rooms
} //impactActive


// CHRIS:  Now we loop through student again
if (impactActive()) {
for(int j=0; j < FileLev3CData.MAX_ROOMS_PER_GRADE; j++) {
  if (!get_is_room_empty(j)) {
    RoomLev3CData room;
    int num_students;
    room = get_room(j);
    num_students = room.get_number_of_students();
    int numGE = 0, numTotal = 0;
    int percentile = 50;
```

```java
            // Loop over students
            // CHRIS: I am finding the number of scores >= my own
            for(int k=0; k < num_students; k++) {
                student = room.get_student(k);
                if (student.mySocioStatus.equals("Popular")) {
                    numGE = strengthPop.tailSet(new Float(student.pPop + 0.001)).size();
                    // System.out.print("STUDENT " + student.get_student_name() + " " + student.
mySocioStatus + " pPop=" + student.pPop );
                    numTotal = strengthPop.size(); }
                else if (student.mySocioStatus.equals("Rejected")) {
                    numGE = strengthRej.tailSet(new Float(student.pRej + 0.001)).size();
                    // System.out.print("STUDENT " + student.get_student_name() + " " + student.
mySocioStatus + " pRej=" + student.pRej );
                    numTotal = strengthRej.size(); }
                else if (student.mySocioStatus.equals("Controversial")) {
                    numGE = strengthCon.tailSet(new Float(student.pCon + 0.001)).size();
                    // System.out.print("STUDENT " + student.get_student_name() + " " + student.
mySocioStatus + " pCon=" + student.pCon );
                    numTotal = strengthCon.size(); }
                else if (student.mySocioStatus.equals("Neglected")) {
                    numGE = strengthNeg.tailSet(new Float(student.pNeg + 0.001)).size();
                    // System.out.print("STUDENT " + student.get_student_name() + " " + student.
mySocioStatus + " pNeg=" + student.pNeg );
                    numTotal = strengthNeg.size(); }
                else {
                    student.statusStrength = 0;
                    continue;
                }

                if (numTotal > 1)
                    // Compute percentiles with me INcluded...
                    // If n=3 and I am in the middle, I should be 50%ile
                    // So remove me from numGE and numTotal and you
                    // have 1/2.  Otherwise, my score is between 1/3 and 2/3.
                    percentile = ((numTotal - numGE - 1) * 100) / (numTotal - 1 );
                else
                    // Judgement call, but if I am the only student in this
                    // status, I think I should be a 2.
                    percentile = 50;

                if (percentile >= 75)
                    student.statusStrength= 3;
                else if (percentile > 25)
                    student.statusStrength= 2;
                else
                    student.statusStrength= 1;
                    // System.out.println(" " + percentile + "%" + " (" + numGE + " of " +
(numTotal - 1 ) + " higher), strength " + student.statusStrength);
            } // Loop - students
        }  // Room is not empty
    } // Loop over rooms
} //impactActive


    // System.out.println("exiting analyze");
    // Set dataAnalyzed so we won't come back thru
    // jt2do: Reset in earlier steps
    dataAnalyzed = true;
} // AnalyzeData

//-============

double pX(double X, double sd, double mu)
{
    // this routine returns the area under the normal dist
    // accurate out to about 6 decimal places
    double tmp1, tmp2, T, Z;
    double rad2 = java.lang.Math.sqrt(2);

    double B1 = .319381530;
    double B2 = -0.356563782;
```

```java
        double B3 = 1.781477937;
        double B4 = -1.8212515978;
        double B5 = 1.330274429;
        double P = .2316419;
        Z = java.lang.Math.abs(X);
    T = 1.0/(1.0+(P*Z));
    tmp1 = T*B1 + java.lang.Math.pow(T,2)*B2 + java.lang.Math.pow(T,3)*B3 + java.lang.Math.
pow(T,4)*B4 + java.lang.Math.pow(T,5)*B5;
    tmp2 = 1.0 - ((0.39894228040143)*java.lang.Math.exp(-0.5*Z*Z)) * tmp1;
        if (X < 0) return (1-tmp2); else return (tmp2);
    }


    double cdf1(double z)
    {
        /*
            Normal distribution probabilities accurate to 1.e-15.
            Z = no. of standard deviations from the mean.
            P, Q = probabilities to the left & right of Z.    P + Q = 1.
            PDF = the probability density.

            Based upon algorithm 5666 for the error function, from:
            Hart, J.F. et al, 'Computer Approximations', Wiley 1968

            Programmer: Alan Miller

            Latest revision - 30 March 1986



            double a-h, o-z;
        */

        double p;

        final double p0 = 220.2068679123761;
        final double p1 = 221.2135961699311;
        final double p2 = 112.0792914978709;
        final double p3 = 33.91286607838300;
        final double p4 = 6.373962203531650;
        final double p5 = .7003830644436881;
        final double p6 = .03526249659989109;

        final double q0 = 440.4137358247522;
        final double q1 = 793.8265125199484;
        final double q2 = 637.3336333788311;
        final double q3 = 296.5642487796737;
        final double q4 = 86.78073220294608;
        final double q5 = 16.06417757920695;
        final double q6 = 1.755667163182642;
        final double q7 = .08838834764831844;

        final double cutoff = 7.071;
        final double root2pi = 2.506628274631001;

        double zabs, expntl, pdf;
        double temp1, temp2;
        zabs = java.lang.Math.abs(z);
//                        System.out.println("In CDF, Z=" + z);
        if (zabs > 37.0)
        {
            pdf = 0.0;
            if (z > 0.0)
                return 1.0;
            else
                return 0.0;
        }

        expntl = java.lang.Math.exp(-0.5 * zabs * zabs);
        pdf = expntl / root2pi;
```

```java
        if (zabs < cutoff)
        {
            temp1 = ((((((((((p6*zabs) + p5)*zabs) + p4)*zabs) + p3)*zabs) +
                    p2)*zabs) + p1)*zabs) + p0;

            temp2 = ((((((((q7*zabs) + q6)*zabs + q5)*zabs + q4)*zabs +
                        q3)*zabs + q2)*zabs + q1)*zabs + q0);

            p = expntl * (temp1 / temp2);

        }
        else
        {
            p = pdf / (zabs + (1 /(zabs + (2.0/(zabs + (3.0/(zabs + (4.0/(zabs +
0.65))))))))));
        }

        //          System.out.println("Out CDF, p=" + p);

        if (z < 0.0)
          return p;
        else
          return (1.0 - p);

    } // cdf1


} // end class GradeLev3CData
```